Tang, Maolin and Lau, Raymond Y.K. (2007) A Parallel Genetic Algorithm for Floorplan Area Optimization. In *Proceedings 7th International Conference on Intelligent Systems Design and Applications*, pages pp. 801-806, Rio, Brazil.

A Parallel Genetic Algorithm for Floorplan Area Optimization

Maolin Tang
Queensland University of Technology
2 George Street, Brisbane, QLD 4001, Australia
m.tang@qut.edu.au

Raymond Y.K. Lau
City University of Hong Kong
Tat Chee Avenue, Kowloon, Hong Kong
raylau@cityu.edu.hk

Abstract

Floorplanning is an important problem in Very Large-Scale Integrated-circuit (VLSI) design automation as it determines the performance, size, yield and reliability of VLSI chips. From the computational point of view, floorplan area minimization is an NP-hard problem. This paper presents a parallel genetic algorithm (GA) for floorplan area optimization. The parallel GA is based an island model with an asynchronous migration mechanism, and is implemented using Web services and multithreading technologies. The parallel GA is compared with a sequential GA that the parallel GA is based on. Experimental results show that the parallel GA can produce better results than the sequential GA when they use the same amount of computing resources. In addition, since the number of islands and migration interval are two important parameters that directly affect the performance of island-based parallel GAs, the impact of the two parameters on the performance of the parallel GA are empirically studied in this paper.

1. Introduction

Floorplanning is an important problem in Very Large-Scale Integrated-circuit (VLSI) design automation as it determines the performance, size, yield and reliability of VLSI chips. Given a set of circuit components, or *modules*, and a net list specifying interconnections between the modules, the goal of VLSI floorplanning is to find a floorplan for the modules such that no module overlaps with another and the objectives are optimized. One of the most important objectives is floorplan area minimization.

From the computational point of view, the VLSI floorplanning problem is NP-hard. Genetic algorithm (GA) is considered to be an effective method for tackling NP-hard optimization problems [5]. GA is a global search technique inspired by evolution. The crux of GA lies in the "survival of the fittest" strategy. It takes an initial set of random *individuals*, termed as the *initial population*. Each individual in the population is a chromosome that represents a solution to the given problem in an encoded form. Using well-defined genetic operators, GA evolves the individuals in the population generation by generation until an optimal or near-optimal solution is found. The fitness of the individuals is evaluated in a *fitness function*.

GAs have been successfully applied on VLSI floorplanning problems [4, 11, 8, 15, 10, 9, 16, 13, 14], some of which are sequential GAs and some of which are parallel GAs. Generally, parallel GAs are intended to speed up the computation. However, parallel GAs may also achieve better results than their corresponding sequential GAs because they are more than a parallel implementation of a sequential GA (their behavior is different from that of sequential GAs). In other words, given the same amount of computing resources a parallel GA may produce better results than its corresponding sequential GA.

This paper presents a parallel GA for floorplan area optimization. This parallel GA is based on a sequential GA for VLSI floorplanning that we proposed in our preliminary research on evolutionary VLSI floorplanning [13]. This parallel GA is based an island model with an asynchronous migration mechanism, and has been implemented using Web services and multithreading technologies. Experimental results show that the parallel GA can produce better results than the sequential GA when they use the same amount of computing resources.

Parameters setting significantly affects the performance of a parallel GA. The number of islands and migration interval are two important parameters that directly affect the performance of island-based parallel GAs. Thus, the impact of the two parameters on the performance of the parallel GA are empirically studied in this paper.

The remaining paper is organized as follows. Section 2 is the problem formulation of the floorplan area optimization problem. Section 3 briefly reviews the related work. Our parallel GA model is presented in Section 4 and the implementation of the parallel GA is discussed in Section 5. Section 6 presents empirical studies on our parallel GA, including the investigation of the impact of the number of islands

0-7695-2976-3/07 $25.00 © 2007 IEEE
DOI 10.1109/ISDA.2007.47

and migration interval on the performance of our parallel GA and the comparison between our parallel GA with the sequential GA. Finally, this research work is concluded and discussed in Section 7.

2. Problem Formulation

A module m_i is a rectangular block with fixed height h_i and width w_i, $M = \{m_1, m_2, \cdots, m_n\}$ is a set of modules. A floorplan F is an assignment of M onto a plane such that no module overlaps with another. In order to minimize the area, a module may be rotated 90 degrees. The area of a floorplan F, $Area(F)$, is measured by the area of the smallest rectangle enclosing all the modules.

Given M, the floorplan area optimization problem is to find a floorplan F such that $Area(F)$ is minimized.

3. Related Work

This parallel GA is based on a sequential GA for VLSI floorplanning that we proposed in our preliminary research on evolutionary VLSI floorplanning [13].

The sequential GA is based on a so called O-tree representation proposed by Guo *et al.* [7]. In the O-tree representation, a floorplan of n modules is represented in a horizontal (vertical) ordered tree of $(n + 1)$ nodes, of which n nodes correspond to n modules m_1, m_2, $\cdots$, m_n, and one node corresponds to the left (bottom) boundary of the floorplan. The left (bottom) boundary is a dummy module with zero width (height) placed at $x = 0$ ($y = 0$). In a horizontal ordered tree, there exists a directed edge from module m_i to module m_j if and only if $x_j = x_i + w_i$, where x_i is the x coordinate of the left-bottom position of m_i, x_j is the x coordinate of the left-bottom position of m_j, and w_i is the width of m_i. In a vertical ordered tree, there exists a directed edge from module m_i to module m_j if and only if $y_j = y_i + h_i$, where y_i is the y coordinate of the left-bottom position of m_i, y_j is the y coordinate of the left-bottom position of m_j, and h_i is the height of m_i. Figure 1 shows a floorplan and its horizontal ordered tree representation.

An ordered tree of n nodes can be encoded in a tuple (T, π), where T is a $2(n - 1)$ bit string identifying the structure of the ordered tree and π is a permutation of the $(n - 1)$ non-root nodes. For a horizontal O-tree, the tuple is obtained by DFS (Depth-First Search) traversing the non-root nodes and edges of the O-tree. When visiting a non-root node, we append it to π. When visiting an edge in descending direction we append an 0 to T and when visiting an edge in ascending direction we append a 1 to T. The horizontal ordered tree shown in Figure 1 is encoded into $(00110100011011, adbcegf)$. We can use the same idea to encode a vertical O-tree.

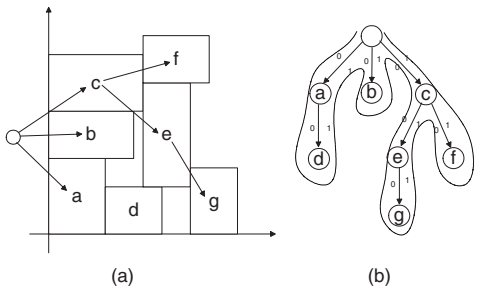

Figure 1. A horizontal O-tree representation and its encoding. In this figure (a) shows a horizontal O-tree and (b) illustrates how to encode the O-tree.

The sequential GA is a steady-state GA, which is used for evolving the sub-population on an island in our parallel GA. This sequential GA runs a fixed amount of time, T, and the size of population is $PopSize$. Below is the outline of the GA.

1. generate an initial population $P(t)$ of size $PopSize$;
2. evaluate the initial population $P(t)$ and find the best individual $best$;
3. while runtime $t < T$
   (a) for each individual in $P(t)$
      i. this individual becomes the first parent p_1;
      ii. select a second parent using roulette wheel selection p_2;
      iii. probabilistically apply crossover and mutation to produce a child c_1;
      iv. use a local optimizer to optimize c_1;
      v. evaluate c_1;
      vi. if c_1 is better than p_1 then use $c_1$to replace p_1 in $P(t)$;
      vii. if c_1 is better than $best$ then $best$:= c_1;
4. output $best$.

Details about the sequential GA, such as the fitness function, genetic operators, selection strategy, can be found in [13].

4. Parallel GA Model

The parallel model used by our parallel GA is an island-based model, which stems from the original research work on parallel evolutionary computation using multiple populations proposed by Bossert [2] and is further developed

by Grefenstette, Tanese, Cantú-Paz *et al.* [6, 12, 3]. In the island-based model, the population is divided into multiple sub-populations, evolving independently from each other on different islands. The sub-populations communicate through migrating individuals from one sub-population to another periodically. This process is called *migration*. The migration is controlled by a communication topology defining the connectivity between the sub-populations, by a migrate rate controlling the number of individuals to migrate, and by a migration interval determining the frequency of the migration.

Different values for parameters, such as the size of subpopulation, crossover probability and mutation probability, can be chosen for each sub-population. Smaller subpopulations are likely to converge faster, but the quality of solutions can decrease. The number of islands, communication topology between islands, migration interval, number of migrants, and selection of migrants, are parameters that characterize the model and they should be correctly set for the appropriate operation of the optimization process. For example, if we need a high selection pressure, we can increase the migration frequency and/or impose the migration of the best individual substituting the worst of the subpopulation.

Often migration in coarse-grained parallel GAs is synchronous occuring at predetermined constant intervals. According to the migration structure chosen, it can increase either, the selection pressure, the diversity or also delay convergence. There is a critical migration rate. Below it, the performance of the algorithm is determined by the isolation of the sub-populations. There are different migration strategies such as to choose emigrants and replace them randomly, or alternatively according to fitness. With a good emigration strategy, it can be obtained a fast convergence and/or a better quality of solutions.

If the topology has a dense connectivity good individuals will spread fast to all the sub-populations and take over the population very fast. If we have smaller dense individuals, they will spread slower and the sub-populations will be more isolated and, will exist a chance to appear different simultaneously individuals. Static topologies specified at the initial and unchanged are used in our parallel GA.

The island model is more than accelerating the computation. A parallel GA developed by using this model has behavioral differences with its corresponding sequential GA. These differences permit to obtain advantages in many situations, inherent not only to the parallelization, but also to the algorithm. This model is adapted better to a computational structure of clusters or multicomputers of distributed memory. If there is no a parallel computer available, this model can be implemented in a network of workstation or in a single processor machine. Figure 2 illustrates the major components and the interactions between them in the parallel GA model.

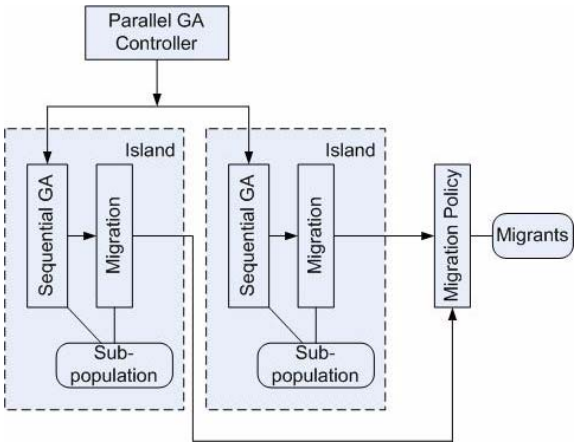

Figure 2. Parallel GA Model

5. Implementation of the Parallel GA

The parallel GA has been implemented using Microsoft Web services and multithreading technologies. Web services is a technology to build application components that can be deployed to the Web and can be used by other applications or components. Multithreading is a programming technique whereby an application can be a program can be divided into multiple asynchronous threads of execution.

Three Web services are built, a sequential GA Web service, a migration Web service, and a parallel GA Web service. Given a floorplan problem, together with genetic algorithm parameters, including the population size, probabilities for crossover and mutation, and selection strategy, the sequential GA Web service produces a floorplan solution. The migration Web service is used by the sequential GA Web service to send its best individual to, and to get the best individuals from, the parallel GA Web service. The parallel GA Web service creates multiple sequential GA threads, and handles the migrations between the multiple sequential GA threads.

The parallel GA system is based on the client-server architecture, with the user application (client) requesting the parallel GA service. The Web services are deployed to a server or different servers and the user application could be run a remote computer.

6. Empirical Studies on the Parallel GA

In this section we use a popular benchmark to empirically study the parallel GA. We will study the impact of the number of islands and migration interval on the perfor-

mance of the parallel GA, and will compare the parallel GA with a sequential GA that the parallel GA is based on.

6.1. The Benchmark

The benchmark used in our empirical studies was *ami33*, a popular MCNC (the Microelectronics Center of North Carolina) benchmark for VLSI floorplanning [1]. The benchmark has 33 modules, 123 nets, 480 pins, and 42 IO pads. The number of modules determines the size of the search space.

6.2. The Experimental Environment

All experiments were conducted on desktop computers with an Intel Core 2 Duo 6300 CPU (1.86GHz) and 2 GB of RAM. The operating system was Microsoft Windows XP.

6.3. Impact of the Number of Islands and Migration Interval on the Performance of the Parallel GA

There are many parameters that may affect the performance of an island-based parallel GA. They include the number of islands, the migration interval, the size of sub-population on different islands, the probabilities for crossover and mutation, and the selection strategy. The size of population, probabilities for crossover and mutation, and selection strategy determine the performance of the sequential GAs running on different islands, and therefore indirectly affect the overall performance of the parallel GA; The number of islands and migration interval directly affect the overall performance of the parallel GA. Considering that the size of population, probabilities of crossover and mutation and selection strategy are not specific to the parallel GA, we focus on studying the impact of the number of islands and migration interval on the performance of the parallel GA.

In order to study the impact, we fixed the size of the population of the parallel GA, probabilities for crossover and mutation, and selection strategy. Thus, the performance of the parallel GA depends only on the number of islands and migration interval. The parallel GA was tested on different combinations of the number of islands and migration interval. The numbers of islands that we tested were 2, 4 and 8, and the migration intervals were 1, 5, 10, and 15.

For each of the combinations we tested the parallel GA on the benchmark in a stand-alone computer for the same computation time, which means all combinations use the same amount of computing resources. Considering the stochastic nature of the parallel GA, we repeatedly tested each combination for 10 times. The size of the population of the parallel GA was fixed to 100. The probabilities for

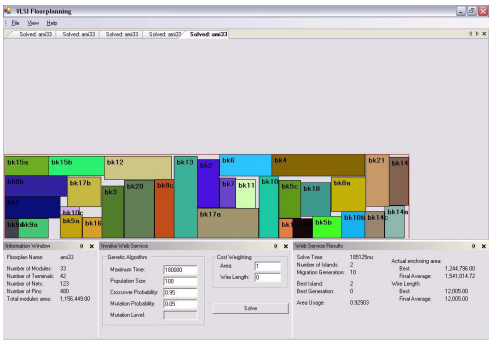

Figure 3. A Screenshot of the Implemented Parallel GA

crossover and mutation were fixed to 0.95 and 0.05, respectively. The computation time was 300 seconds for each of the combinations per run. For example, when the number of islands was 4 and the migration interval was 5, the parallel GA created 4 islands, each of which having 25 individuals. All the islands were allocated to the same computer (The islands could be allocated on different computers. However, in order to make sure each run consumed the same amount of computing resources, we allocated the islands on the same computer.). Each of the islands sent its best individual to, and got the best individuals from, the other islands every 5 generations. Figure 3 is a screen of one of the experiments.

Tables 1-3 show the experimental results. Table 1 shows the statistics for the experiments for 2 islands with different migration intervals (1 generation, 5 generations, 10 generations and 15 generations); Table 2 displays the statistics for the experiments for 4 islands with the four different migration intervals; Table 3 gives the statistics for the experiments for 8 islands with the four different migration intervals.

It has been seen from the experimental results that different combinations resulted in different performances of the parallel GA. Among the tested combinations of the parameters, the combination, number of islands = 2 and migration interval = 5 generations, gave the best performance on average. The combination, number of islands = 4 and migration interval = 1 generation, created the worst performance on average.

In addition, it has been observed from the experimental results that the correlation between the number of islands and migration interval. When the number of islands is smaller, the migration interval should be shorter in order to get the best performance, and when the number of islands is larger, the migration interval should be longer in order to get the best performance. For example, when the number of islands was 2, the parallel GA generated the best result on average when the migration interval was 5 generations;

Table 1. Statistics for the experimental results on the benchmark (2 islands)

migration interval	best	mean	std dev
1 generation	0.96694	0.936119	0.01374
5 generations	0.95369	0.938156	0.00926
10 generations	0.93968	0.933279	0.00444
15 generations	0.94533	0.932369	0.00760

Table 2. Statistics for the experimental results on the benchmark (4 islands)

migration interval	best	mean	std dev
1 generation	0.93388	0.925533	0.00458
5 generations	0.94080	0.931818	0.00589
10 generations	0.94533	0.935849	0.00826
15 generations	0.93804	0.928595	0.00633

Table 3. Statistics for the experimental results on the benchmark (8 islands)

migration interval	best	mean	std dev
1 generation	0.94442	0.927186	0.01154
5 generations	0.94155	0.931792	0.00749
10 generations	0.95489	0.933897	0.01314
15 generations	0.95551	0.935583	0.01083

when the number of islands was 5 generations, the parallel GA created the best result when the migration interval was increased to 10 generations; and when then the number of islands was 8, the parallel GA found the best result when the migration interval was further increased to 15 generations.

In the experiments the population size was fixed. Therefore, the smaller the number of islands, the larger the sub-population was and therefore the more diverse it would be. Hence, a sub-populations did not need to be diversified by immigrating individuals from the other sub-populations frequently. However, when the number of islands became larger, the size of sub-populations became smaller and therefore the diversity of sub-populations became poorer. Thus, it was necessary to more frequently diversify the sub-populations by increasing the frequency of the migration.

6.4. Comparison with the Corresponding Sequential GA

An island-based parallel GA is more than a parallel implementation of a sequential GA. The behavior and performance of an island-based parallel GA might be different from that of its corresponding sequential GA.

The corresponding sequential GA is a special case the parallel GA where the number of islands is 1. In order to make sure that the comparison was fair, we tested the sequential GA on the same benchmark. We run the sequential GA for the same amount of time (300 seconds) for 10 times and recorded the experimental results. Table 4 shows the statistics for the experimental results.

Table 4. Statistics for the experimental results for the sequential GA on the benchmark

best	mean	std dev
0.93934	0.926202	0.07786

It can be seen from the experimental results that the parallel GA produced better floorplanning results than its corresponding sequential GA for all the tests except for the test when the number of islands was 4 and the migration interval was 1 generation. When the number of islands and migration interval were well selected, the performance of the parallel GA was significantly better than that of the sequential GA. For example, when the number of islands was 2 and the migration interval was 5, the area usage was 0.938156%, which is 1.2% better than that of the sequential GA. Thus, the selection of number of islands and migration interval are very important to the parallel GA.

7. Conclusion and Discussion

This paper has presented a parallel GA for floorplan area optimization — a challenging combinatorial optimization problem in VLSI design automation. This parallel GA distinguishes from existing parallel GAs for VLSI floorplanning in that it adopts an island model with an asynchronous migration mechanism, and uses a different chromosome representation and genetic operators.

Compared with commonly used synchronized migration mechanisms, the asynchronous migration can make better use of computing resources when the parallel GA is deployed to a distributed computing environment where the performances of processing units are different. In synchronized migration mechanisms, faster processing units must wait for slower processing units. As a result, some valuable computing resources of the faster processing units are wasted. In contrast, in our asynchronous migration mechanism faster processing units do not wait for slower processing units. The asynchronous migration has also potentials to make the parallel GA more efficient. For a parallel GA using synchronized migrations, its efficiency depends on the performance of the worst processing unit. In contrast, the efficiency of our parallel GA depends on the overall performance of the processing units.

In this paper we have also conducted an empirical study on the impact of the number of islands and migration in-

terval on the performance of the parallel GA. The study reveals that the performance of the parallel GA is sensitive to the the number of islands and migration interval settings. The study also reveals that there are some correlations between the number of islands and the migration interval. When the number of islands is large (the sub-population size is small), the migration interval should be shorten in order to keep the sub-populations diverse; when the number os islands is small (the sub-population size is large), the migration interval should be lengthen as the diversity of the sub-populations would be a problem and therefore there is no need for the sub-populations to exchange individuals. In fact, frequently exchanging individuals between the sub-populations may lead to communication overheads between the processing units.

To determine the number of islands and the migration interval is a time-consuming task. Thus, a good solution to the problem would be to make the parallel GA adaptive. This could be done in two ways. One is to equip the parallel GA with a process that can determine the parameters setting before the evolution actually commences. Another would be to make the parallel GA itself adaptive - the parallel GA can dynamically change the parameters during the runtime in the light of the status of the evolution.

This paper has also compared the performance of the parallel GA with a sequential GA that the parallel GA is based on. When the number of islands or the migration interval are not set correctly, the parallel GA may create worse results than its corresponding sequential GA. However, when the number of islands or the migration interval are set correctly, the parallel GA can produce significantly better results than the sequential GA.

The work presented in this paper represents our first effort towards parallel GAs for VLSI floorplanning. There are many interesting issues needed to be further investigated. For example, the migration topology used in the parallel GA is a fully connected one, which means significant communications between the sub-populations. How to reduce the migration between the sub-populations without compromising the quality of solutions would be an interesting research problem that we will study in the future.

8. Acknowledgement

The authors would like to thank Alvin Sebastian for his contributions to the implementation of the sequential GA, and Aaron Moore for his contributions to the implementation of the parallel GA.

References

[1] The mcnc website, http://www.mcnc.org.

[2] W. Bossert. Mathematical optimization: Are there abstract limits on natural selection? In P. S. Moorehead and M. M. Kaplan, editors, *Mathematical Challenges to the Neo-Darwinian Interpretation of Evolution*, pages 35–46. The Wistar Institute Press, Philadelphia, PA, 1967.

[3] E. Cantu-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[4] J. Cohoon, S. Hegde, W. Martin, and D. Richards. Distributed genetic algorithms for the floorplan design problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(4):483–492, 1991.

[5] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company, Inc., Reading, MA, 1989.

[6] J. Grefenstette. Parallel adaptive algorithm for function optimization. Tecn. Rep. CS-81-19, Vanderbilt University, Nashville, TN, 1981.

[7] P.-N. Guo, T. Takahashi, C.-K. Cheng, and T. Yoshimura. Floorplanning using a tree representation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(2):281–289, 2001.

[8] B. Gwee and M. Lim. A GA with heuristic based decode for IC floorplanning. *Integration, the VLSI Journal*, 28(2):157–172, 1999.

[9] C.-T. Lin, D.-S. Chen, and Y.-W. Wang. An efficient genetic algorithm for slicing floorplan area optimization. In *Proc. IEEE International Symposium on Circuits and Systems*, volume 2, pages 879–882, 2002.

[10] S. Nakaya, T. Koide, and S. Wakabayashi. An adaptive genetic algorithm for VLSI floorplanning based on sequence-pair. In *Proc. IEEE International Symposium on Circuits and Systems*, volume 3, pages 65–68 vol.3, 2000.

[11] M. Rebaudengo and M. Reorda. GALLO: A genetic algorithm for floorplan area optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(8):943–951, 1996.

[12] R. Tanese. Parallel genetic algorithms for a hypercube. In J. J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*. Lawrence Erlbaum Associates, Publishers, 1987.

[13] M. Tang and A. Sebastian. A genetic algorithm for VLSI floorplanning using o-tree representation. *Lecture Notes in Computer Science*, 3449:215–224, March 2005.

[14] M. Tang and X. Yao. A memetic algorithm for VLSI floorplanning. *IEEE Transactions on Systems, Man and Cybernetics, Part B,*, 37(1):62–69, February 2007.

[15] I. Tazawa, S. Koakutsu, and H. Hirata. An immunity based genetic algorithm and its application to the VLSI floorplan design problem. In *Proc. IEEE International Conference on Evolutionary Computation*, pages 417–421, 1996.

[16] C. Valenzuela and P. Wang. VLSI placement and area optimization using a genetic algorithm to breed normalized postfix expressions. *IEEE Transactions on Evolutionary Computation*, 6(4):390–401, 2002.